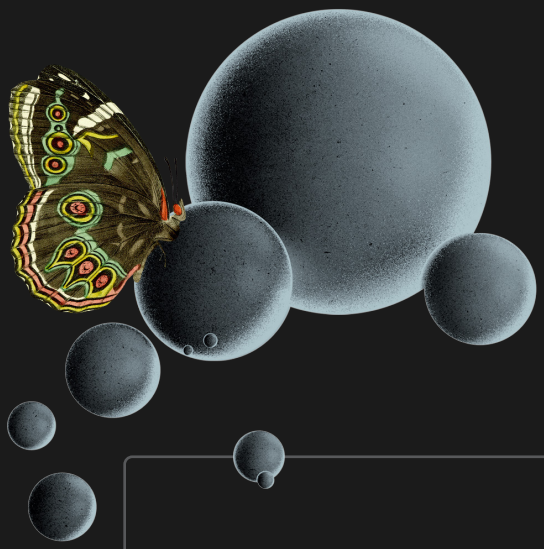
An Argument for Continuous Security Testing

Forming the foundation of your security validation program



When completing a Ph.D. program, doctoral candidates are required to submit a thesis argument. The argument represents a central claim put forward, as the foundation of their research. It serves as the main focus or guiding principle, often representing the last few years of the student's work. The argument typically presents a unique perspective, theory, hypothesis, or solution to a problem that has no universally-accepted answer.

Throughout the thesis, the doctoral candidate will present and discuss their findings, analyze the data, and provide arguments and evidence to support or refute their initial thesis argument. The conclusion of the thesis should summarize the main findings and evaluate the extent to which the original argument has been successfully addressed or resolved.

Here, we will follow a similar, albeit briefer, approach to introduce continuous security testing (CST) as a greater topic. The argument we will be making today is this: continuous security testing should act as the foundation of your security validation program.

Outline

01 Introduction

A fictitious example

Let's say it's your first day on the job. You just accepted a post as the Chief Information Security Officer (CISO) of a mid-sized basket-weaving business, a thriving enterprise in our fictional world. Your new company, Baskets For All, employs hundreds of basket-weavers, two call centers, a robust sales team, a team of designers and engineers, a research and development lab, an IT department, and an executive team. What is the first thing you do?

You probably start by reviewing your security posture. What assets are in the inventory? What controls are in place? What is the composition of the IT team? What processes are in place for incident response and vulnerability management?

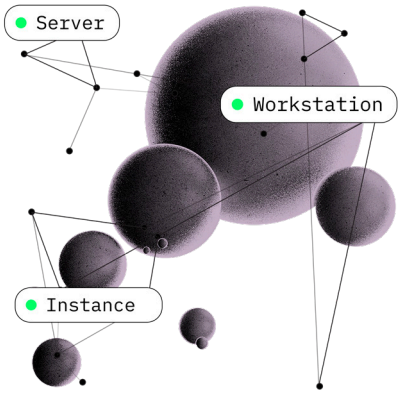
This discovery will provide a picture of what exists today. Let's say it uncovers:

- **2,500 workstations / laptops; a mixture of Windows and MacOS**
- **300 Linux virtual machines running inside AWS**
- **A dynamic Kubernetes cluster, also deployed in AWS, consisting of up to 50 containers**

You see that all workstations and virtual machines are running the latest in-fashion Endpoint Detection & Response (EDR) agent, along with a SIEM logging agent. The EDR isn't running on the containers, but they are running the logging agent. The containers are especially locked down, with limited system binaries installed. Inside of AWS, you are running an off-the-shelf Intrusion Detection System (IDS), for inspecting network traffic, and a Web Application Firewall (WAF) for robust firewall protection.

When you went over the security processes, you learned that the company contracts out for quarterly red team assessments. There is also a top-of-the-line vulnerability scanner that runs daily and a process for resolving any discovered vulnerabilities that have a CVSS score of above 7.0. Digging through the numbers, you see that 98% of vulnerabilities are patched within 72 hours - an astonishing statistic!

But you also know that Baskets For All just endured a cyber attack, which was the reason they hired you in the first place. With all of this security, how could a breach have happened?



Background information

Why you need security testing

A security strategy should revolve around "the endpoint". An endpoint could be a laptop, workstation, server, container, virtual machine, traffic light, camera, WiFi-enabled toothbrush,... basically, if the device is networked and runs code - it's an endpoint.

Defenses, such as EDR, are installed on endpoints to prevent malicious activity, whether static or dynamic. This is especially important on user-controlled devices, such as laptops. As is often stated, people are the biggest vulnerability to any organization - so running a protective agent 24x7 on their computers is a great first-layer defense.

This poses two central problems so far:

1. **Endpoint defenses aren't compatible with all endpoint operating systems**
2. **Endpoint defenses don't work as expected all of the time**

For the first problem, you can look at any environment and explicitly determine which devices match the profile of a supported endpoint defense. In the case of Baskets For All, all but the containers had supported EDR agents running. This is a pretty common case to encounter in the real world. But because of the straightforward solution - install EDR on all supported endpoints and accept the risk on the others - this won't be the focal point in this white paper.

For the second problem, the only solution is to validate your defensive efficacy through security testing.

Security testing is an afterthought. Most organizations establish their people. Their product. Their technology stack. Their cyber defenses. And last and intentionally least (usually due to compliance requirements) their security validation process. Security is thought of as a sunk cost - the cost of doing business - with security testing at the bottom of the barrel.

Over the years, security testing has seen a progression of strategies. Other writings have covered the most popular of the bunch, from pentesting to Breach and Attack Simulation (BAS). Each has brought its fair share of pros to the table - and a measurable number of cons. While much broader in overall scope, at its core security testing aims to understand if your defenses work.

Research problem

Why your testing should be continuous

No matter the strategy, security testing has always been point-in-time. When you run a security assessment, you plan, execute and report on the results. All of these are very timeline-centric: there is a distinct start and end to each phase of this process.

The problem is that technology moves too fast.

An EDR, and the operating system it's installed on, is under constant iteration. The programs are being updated. Features are being added. The pace of software development is that of a sprint, not a marathon. Each change brings the risk of a new vulnerability or a blind spot in the defensive agent, which hasn't created a signature for a new malicious sequence of behaviors.

This gets even worse at scale, as changes to the environment bring additional variables to the table. A laptop may move subnets, as it (physically) moves from the corporate office to the home office. Or a mass software installation directed from a Mobile Device Management (MDM) portal may not complete the installation of all targeted machines, leaving some endpoints in an awkward state.

All of this means security testing has to pick up the pace. It needs to graduate from point-in-time to continuous.

Consider a convenience store. The store is small in surface area, maybe a few hundred square feet on average. But protecting it isn't foolproof; there's a margin of error involved.

It's not that there are no defenses. One could argue that the defenses are overkill for the environment and the "crown jewels" being protected. The store is armed with a security system, including active cameras 24/7 and alarms for when it's locked up at night. And of course, there's a cashier situated right next to the door.

If this security were applied to a computer network, you'd feel pretty good about it: the cameras are the event logs, detecting all activity happening in the space. The alarms represent the EDR, detecting and responding to malicious actions. The cashier is the SOC, a human-in-the-loop offering contextual assistance.

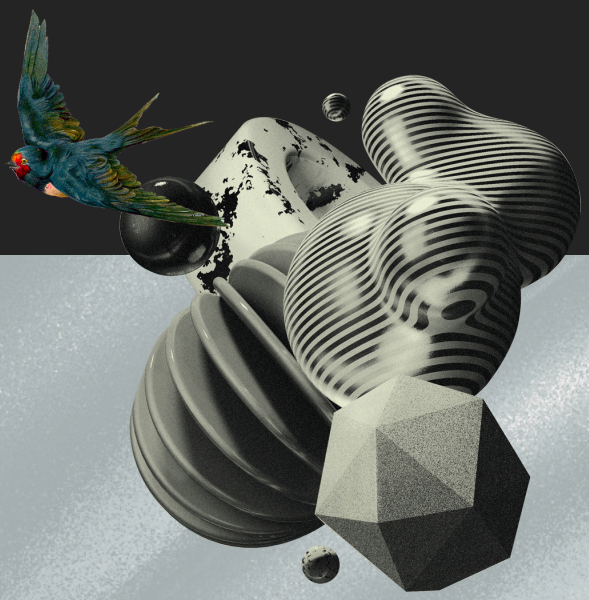
But the convenience store can still be robbed. Despite the restricted space and enhanced protections, there is no physical way - that is also justifiable cost-wise - for the store to eliminate the risk completely. Shoplifters will simply fly under the radar.

If this is the case for a single convenience store with a single physical location - imagine how much more true it is on the Internet, where every person on the planet could walk through your door at the same time.

02 Thesis statement

Continuous security testing should form the base of any security validation strategy.

The rest of this white paper will outline the argument for this statement. While other forms of validation can play a supporting role, if you can only do one form of testing, CST should be it.



03 Competitive review

The closest we've come to CST today is BAS, described in depth elsewhere. BAS, as an industry, promises to automate the value of a red team through a solution with less efficacy but one that can be run every day.

We're not here to argue the value of BAS, or whether it fulfills its promise, but let's contrast it with CST, in relation to this thesis argument.

Promise statement

The promise of CST is to *"know with certainty if your defenses will protect you against emerging threats."* This is the most telling departure from the BAS approach. Ask yourself this: does your red team tell you with certainty that your defenses will protect you from emerging threats? The answer is most definitely no, as the focus of red teams is too narrow and the scale on which they operate too low to answer a question this grandiose. Even if a BAS solution were to fulfill its promise - which is to automate a red team - to full efficacy, it wouldn't answer the question CST addresses.

If your goal is to have complete security, and we can agree that the endpoint is where you should focus, then it's logical to put your defensive energy (EDR) on the endpoints. Therefore, logic would dictate that security testing should focus on the value and efficacy of that defensive solution, which is what CST is purpose-built to do.

Red teams, and the automation of their processes, would act as a cherry on top; they should be part of an advanced security stack instead of a foundational requirement for everyone.

Everything in security is testing

I often say that cybersecurity is actually composed of around 15 sub-industries. There is offensive security, the topic of this book. But there is also incident response, malware analysis, blue teaming, purple teaming, SOC, attack surface reduction, cyber threat intelligence, ... and several more.

Most sub-industries are actually a form of security testing.

Consider malware analysis. The painfully methodical process of working JUMP statement by JUMP statement through the assembly code of a binary is a form of testing. The goal is to extract the behaviors of a specific piece of malware in order to signature it for a defense to catch it, which in turn must be validated (for example, through a YARA rule).

Each sub-industry of cybersecurity exists to fuel one of two engines: defense or validation. The effort is either actively defending an endpoint, or it is validating that the defense is working.

You might say, *"Ah, but testing from the endpoint is not enough! I must also test the network, my detection engineering rules, my VPN, my..."*.

But let's test that theory a bit.

Consider two scenarios:

1. **If your Web Application Firewall (WAF) works 100 percent, are you protected?**
2. **If your WAF works 0 percent but every endpoint in your environment stops all malicious behaviors - including insider threats - are you protected?**

Which scenario is correct? The second one, of course.

Let's make it more extreme. Let's say every endpoint in your environment has every known vulnerability (whoa!) - but it still prevents all exploit attempts and blocks all malicious behaviors. Are you still protected? The answer is, amazingly, still yes.

What does this tell you? Endpoints are the center of your infrastructure. If you can protect every individual one, you have achieved a level of security in the 100th percentile.

This doesn't mean you should drop your WAF. You should still have one and configure it reasonably - but it's not as important as your endpoint defense. A WAF is remote and disconnected from the endpoint, so it is a secondary concern.

Defensive interoperability

Remember the hot wash meeting, where a red or purple team goes over the findings of their security assessment and provides the defenders with a list of things to fix? BAS accelerates this by injecting automation. Instead of a point-in-time approach, BAS aims to generate and keep this list of "honey-do's" always up-to-date.

But that hardly solves the problem.

In the mid-2010's, this seemed like the right direction for security testing; the concepts of modeling, intelligence and scale had yet to be addressed. The automatic honey-do list overlooked one essential factor, which is easy to pick on in retrospect: security teams are strapped for resources already and by giving them an endless number of new things to do, you are effectively creating hopelessness. Teams avoid tools that do this because, as security is already considered a sunk cost, does someone really want a solution that has no practical end state? It's almost like running a marathon where the finish line keeps being pushed back 5 miles. There has to be an end state.

Humans need a finish line or a goal to shoot for because it provides a sense of purpose, motivation, and direction. Having a clear objective or endpoint gives individuals something to strive for and can drive them to work harder, push their limits, and achieve their desired outcomes.

CST adopts the view that - for security testing to be workable - it needs interoperability with the defensive tools running on the endpoint. This takes the form of self-healing where any attack scenario that uncovers an unprotected behavioral sequence, or an undetected file signature, is automatically sent to the EDR for resolution.

A core CST belief is that customers shouldn't bear the responsibility of fixing/securing broken promises from defensive vendors. For example, if an EDR promises to stop all breaches - and a security test finds an attack sequence that isn't prevented - the responsibility to fix the issue is on the vendor, not the customer who purchased it. And this should happen automatically, preferably in a timely manner.

The industry has accepted the opposite for far too long.

Multiple, often overlapping, endpoint defenses are purchased - more as an insurance policy than to boost the odds of protection in a breach. And even then, companies which don't specialize in cybersecurity are staffing hundreds of security professionals. We should ask ourselves, exactly why does a company that specializes in home improvement projects employ a threat intelligence unit?

We accept that our defensive vendors aren't working as expected - but we still have a business to run - so we bear the responsibility. Why don't we stop purchasing these tools, until they provide transparency into their efficacy? If we do, when we inevitably get breached we have little political fallback. Our purchased tools act as an insurance policy for our own jobs. We can effectively say to the board of directors: but we bought all these tools! Or if we were trying to get out of a personal relationship: it's them, not us.

Security testing should accomplish two things:

1. **Establish a shared reality about the security posture of an organization. Everyone, from security engineers to the board of directors should sing from the same hymn book.**
2. **Automatically resolve any security issues found. Manual resolution of security bugs is quickly becoming a legacy process. Testing needs to both find issues and engage the defense on a resolution.**

Scale

As BAS solutions are designed to automate a red or purple team, they are naturally confined to the same scope as their manual counterparts: testing on a small n-size of development endpoints. The results that follow are extrapolated to the environment as a whole, as a way to reduce the risks that would otherwise occur from running the unstable TTPs that offsec teams pull out of their hats. But as pointed out in other content, extrapolation is not an effective way to measure the efficacy of your controls at scale because, by definition, it is not testing your controls at scale.

Development or production?

When you red team, what you're really doing is uncovering your unknown security holes. Red teaming is, by definition, the process of playing devil's advocate. So you're poking and prodding your systems in unintended ways - the same ways an adversary does - hoping to uncover a weakness.

This doesn't come without risk. If you poke your system too hard, it may topple over. If you do this in production, your customers may feel the weight of the fall.

But consider this: if you get attacked in the middle of the night by an attacker using the same techniques as your internal red team, you'll still topple over, except this time you won't have control. Worse, you may have compromised your customers. In other words, by trying not to topple your system, you may be in for an even bigger fall.

Most organizations have multiple environments. There is a development environment for engineers to build solutions. The QA environment for testers. The staging environment to mirror production, just in case you need it. And finally, the production environment to serve the public.

"If my staging environment mirrors production, isn't that the same as testing prod?"

If you find yourself asking this question, consider if your staging environment is exactly the same as production. Your staging environment is likely a test bed to flush out bugs that are hard to debug in production. Most staging environments run the same software as their production counterparts but the scale, network rules and infrastructure is far different, to lessen the cost.

Why do you think organizations with top-dollar defensive tools get hacked every day?

You should expect quality systems at your organization. Ones that do not topple over during a red team assessment. Your systems should be resilient to testing and get stronger over time. Your engineers will respect this and you'll be doing more long term good - for you and your customers.

When using CST against a production environment, the harmful effects of a real adversary are not conducted. There are several safety precautions taken, such as not encrypting files during a ransomware attack but instead copying a file and encrypting the copy, to prove it is either possible or not without defensive prevention.

Now for the question of autonomous systems.

Manual red teams rarely get posed the 'can I do this in production?' question. People are wary of automated processes, especially those built for offensive security. But this reluctance will change. It will take time. It will take effort. Most of all, it will take reforming how you think about your security.

Red teaming is great but the time constraint and cost make it unachievable for most companies. For those remaining, running several red team exercises a year just isn't enough. It's better than none, but continuous testing is better. And the future. The only way to get there is to embrace autonomous red team software.

Okay, you get it. You're technical, you see this as the optimal strategy. But how do you get your manager, the one who takes the blame when things go south, to see this as well?

Move into the area slowly. Instead of going full-throttle and deploying an autonomous testing solution into your network, use a tool that allows you to have full manual control. Do your first several security assessments without turning on any of the autonomous bells and whistles. Plan them in small doses. Not every assessment needs to be a full-blown endeavor. Then, over time, it will become easier to introduce autonomy as you build trust, both with your organization and with your tool of choice.

In the end, choosing autonomous testing means choosing the future. Your attackers are using them for their efficiency in hacking you, so you should be prepared to defend yourself with a dose of autonomy yourself.

In contrast, CST is designed to run on production, against every endpoint you have. This provides exact observability around defensive efficacy, bypassing the sharp edge cases of the extrapolated approach completely.

Furthermore, CST is designed to run where endpoint defenses cannot. Earlier, I mentioned two central problems that running an EDR can present:

1. **Endpoint defenses aren't compatible with all endpoint operating systems**
2. **Endpoint defenses don't work as expected all of the time**

We addressed the second but mostly skipped past the first.

The fact that endpoint defenses are not compatible with all endpoint operating systems doesn't mean you should convert all of your containers to Windows servers or downgrade your software to match the limitations of a vendor. In fact, it's quite the opposite.

CISA, the U.S. government agency tasked with building a more secure, resilient infrastructure for the future, writes on their site:

"When security experts give cybersecurity advice, they usually assume you are only willing to make small changes to your IT infrastructure. But what would you do if you could reshape your IT infrastructure? Some organizations have made more aggressive changes to their IT systems in order to reduce their "attack surface." In some cases, they have been able to all but eliminate (YES, WE SAID ELIMINATE!) the possibility of falling victim to phishing attacks."

They recommend doing this through a replacement of insecure devices with secure ones:

"While all operating system vendors work to continuously improve the security of their products, two stand out as being "secure by design," specifically, Chromebooks and iOS devices like iPads.

Some organizations have migrated some or all their staff to use Chromebooks and iPads. As a result, they have removed a great deal of "attack surface," which in turn makes it much harder for attackers to get a foothold. Even if an attacker were able to find a foothold on those systems as part of a ransomware attack, the data primarily lives in a secure cloud service, reducing the severity of the attack."

The problem is this shift is too gargantuan a task for most organizations. In fact, the larger the organization, the more challenging it is to heed this advice.

This is where an EDR comes in handy. If swapping out the operating systems on your endpoints is a 5 year plan, you need an actionable strategy for today. An EDR running on your endpoints can adequately protect against most off-the-shelf attacks - but only if correctly configured and continuously validated.

More on secure devices later.

Transparency

Enterprise security solutions, BAS included, enjoy hiding behind a veil of secrecy. The industry as a whole has succeeded at using this technique to "gate" people who don't understand the complex world of cybersecurity. You can see this most clearly by analyzing what is open vs what is closed in any particular solution. If every component in the product is closed-source - there is no visibility into the inner workings of the system. It's a black box.

Aside from the lack of transparency, customers of these solutions face another downside: if they switch vendors they lose access to their prior work. For example, if I purchased a BAS testing solution last year and my internal security team uses it to write 100 custom tests - specific to my company's use case - if I leave the vendor in search of one with a richer feature set, I may lose my custom work. This lock-in effect means I would have to start all over again with the new vendor.

In CST, the philosophy is that all code running on your endpoints must be open whereas the code running on a vendor's side can be open or closed.

This simple philosophy empowers customers to have complete visibility and enhanced expectations. Want to dig deeply into the code of a test or endpoint agent? Go for it. Want to pack up your tests and move to any CST solution? Great! Tests should be viewed as a commodity, forcing testing solutions to stand out for proprietary feature sets that sit within their closed-source offering, such as intelligent scheduling of tests or a rich data visualization experience.

To read more...

I have documented methodology and the results from a real continuous security assessment. To read these, and the rest of Irreducibly Complex Systems, head to https://www.yellowduckpublishing.com and pick up a copy today.

04 Discussion

Address counter arguments

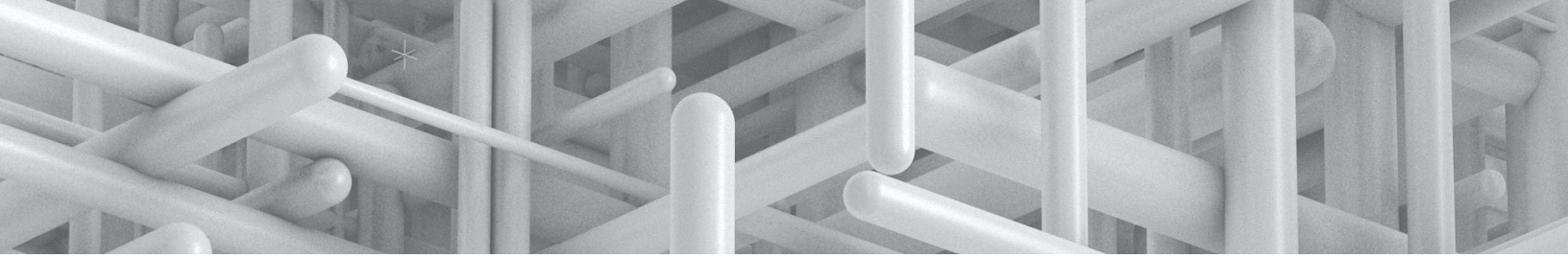
While I feel comfortable with the confirmation of the thesis presented here, there are counter arguments worth considering. I will try to fairly assess the primary arguments.

Only humans can accurately perform a security assessment

The argument here is that only humans can contextualize high-risk scenarios during a security assessment, as a computer cannot balance its effectiveness with the odds of being detected, for any given action. This is mostly true, if comparing apples-to-apples. However, our thesis does not state that CST should replace a manual red team, only that it should be the foundation of a validation strategy. Building upon this strategy with manual red teaming will bolster the security posture of any organization.

People won't deploy another agent across their infrastructure

This is not an argument against the concept of CST but more so the practical ability to get it off the ground. CST institutes several architectural designs to alleviate agent-fatigue and make the process as easy as possible. Agents, called probes, are usually only a few kilobytes and can be installed with a single command. They also run as regular users, not administrators, and use a non-measurable amount of system resources. Ultimately, the value proposition of CST has to outweigh the friction of deploying agents, regardless of how easy it is, and only the end user can make that determination.



The agent needs to be part of the security assessment

CST makes a departure from other security strategies, such as red teaming or BAS, which dictate the agent running the tests should be part of the assessment. This would mean the agent needs to be evasive and avoid detection and quarantining by an EDR. This naturally limits the number of agents that can be active in an environment, as otherwise the presence of a C2 network would be easily squashed by the defense. At first glance, this would disqualify CST as a valid testing strategy; but CST instead considers evasive agents as an advanced manual practice.

Probes are encouraged to be installed as a service and are allowed to run - unimpeded. Probes spin each test into a separate process, which is what the defensive reaction is judged against. This allows CST to run at high-scale: the probe is guaranteed to be active and only the child processes can be quarantined and evaluated. This separation allows the probe to monitor the defensive reaction to a specific sequence of behaviors versus the probe itself getting signatured based simply on its network activity or static state.

You can run tests on a small number of endpoints

Hopefully, the thesis argument in this white paper was enough to knock this fallacy down. Small variations in the behavior of tests or the state of the computer (or larger infrastructure) can create variations in how endpoint defenses react. This can be demonstrated a number of ways, including an example from previous writings. In short, I demonstrated a popular strategy of raising CPU usage on an endpoint to force an EDR to drop packets, as it attempts to safeguard against using too many system resources. This built-in safety mechanism has many variations, some known but many unknown (due to the complexity and closed-source nature of endpoint defenses; some EDR products contain several million lines of code).

If you want to achieve the CST promise, know with certainty if your defenses will protect you from emerging threats, you simply need to test on all your endpoints.

A test implementation can be tweaked to evade the defense

This argument carries the most weight - and most testing strategies squirm when trying to answer it. Working at MITRE, while ATT&CK was gaining prominence, one thing stood out about the purple teams embracing it: they treated the matrix like a Bingo card. Their goal was to put a green (protected) dot on each one of the technique boxes. In fact, I commonly refer to this as “ATT&CK Bingo”!

The ATT&CK matrix consists of tactic columns and technique rows. The tactic columns represent the high-level objectives or goals of an attacker. The technique rows provide detailed information about the specific methods employed by attackers within each tactic.

ATT&CK intentionally veers away from including the “P” (from TTP) into the matrix because of the infinite number of procedural variations existing within a technique. This didn’t stop purple teams from assuming that if they could protect against a single procedure in a technique, they were protected from the entire technique. This is, of course, a flawed attempt to display protection in a quantitative way - but it should highlight just how hard the problem is.

Does this make you feel safer?

CST tries to tackle this through constant variation. By running a never-ending supply of tests, every day, and across all endpoints, the idea is you will eventually run a statistically significant number of variations. This process, not dissimilar to the Law of Large Numbers (which states that as the number of independent trials or observations increases, the average of those outcomes will converge to the expected value or true probability of the event), attempts to provide a quantitative numerator to a problem with an infinite denominator.

This is not without limitations, which will be covered later in the next subsection.

You can run security tests without being on the endpoint

This is a favorite argument from the world of "agentless" solutions, which scan endpoints from the outside instead of running an internal process on each.

For example, an agentless antivirus solution utilizes network-based scanning techniques to inspect incoming and outgoing traffic without requiring software agents to be installed on every endpoint. Similarly, agentless vulnerability scanners assess the security posture of systems by utilizing protocols like SNMP (Simple Network Management Protocol) or SSH (Secure Shell) to collect information about the target devices.

While agentless solutions offer benefits such as simplified deployment and reduced resource consumption, they have a specific limitation: they rely entirely on the accessibility of target systems. Endpoints run firewalls and other protective measures to thwart introspection from outside processes, limiting the reach of agentless solutions.

This leaves them with a read-only vantage point from the perimeter of a device, which can be useful in vulnerability management or asset inventory, but quite impossible to fulfill the CST promise to know with certainty if your defenses will protect you against emerging threats.

CST does not cover advanced security scenarios

A common argument against automated solutions is that they don't provide coverage against complex, multi-stage attack scenarios. The most common example is dumping credentials on a computer and attempting to use the discovered passwords to laterally move to another endpoint, where the exercise can be repeated until exhaustion.

This sequence of behaviors is better served for manual red team assessments or the small n-size of BAS.

The reason is simple: these scenarios are about information-gathering, not testing the efficacy of the defense. A valid CST test may conduct this same multi-stage attack but it will stop short of invoking a new probe on a remote endpoint. Instead, it will dump credentials and use those credentials to test whether the defense stops it from trying lateral movement.

CST is not concerned with building an attack graph of all possible network hops from one machine to another but instead - per endpoint - reports whether the local defense can stop the behavior when induced.

The same analogy applies to vulnerabilities: it's not enough to know an endpoint has a vulnerability - you need to know if that vulnerability is exploitable. An EDR is capable of blocking exploit attempts, which is the reason you have it. An unexploitable vulnerability is the same as a lateral movement sequence that cannot be completed due to defensive interference - it's theoretical.

Limitations

Like most solutions, CST is not a silver-bullet. It comes with limitations. We covered one in the previous section, *"A test implementation can be tweaked to evade the defense"*, which is less a limitation on the approach CST takes and more a limitation on security validation as a whole. I call this the "many variations problem". CST tries to neuter this limitation through probabilities but that, by definition, is not an iron-clad approach, only a best-effort.

Another limitation falling under this line of reasoning is the expansive surface area of an endpoint. This is the focus of other content, so I will be brief here, but it goes like this: common operating systems provide the user with a tremendous amount of control and no clear boundaries between processes, applications, memory and other OS-level components. This combination of factors means the endpoint has many targetable attack vectors - or a very wide surface area.

Combining the wide surface area with an unpredictable user behind the keyboard makes defending these endpoints difficult. EDR's must account for the innumerable technical sequences possible on the device while simultaneously monitoring how a user behaves on it. The hard part comes when trying to separate the malicious sequences from the sea of benign ones, a true needle in a haystack scenario.

Testing strategies, like CST, are great at poking holes in defenses because of this problem. It is a defensive limitation, not a testing one.

However, it's the duty of CST to accurately - and continuously - rediscover this limitation as applied to emerging threats and autonomously resolve them. Defenses have been dealt a tough hand, trying to detect and prevent malicious activity across such an expansive surface area (made more difficult by having limited system resources to expend on the device). The best approach you can take is to establish an automated cat-and-mouse testing system where issues are found - and fixed - automatically.

Reducing surface area by changing your underlying operating systems is the ultimate solution - a topic I've written about extensively.

Implications

Security testing is all about the resolution: what can you infer from the results?

Previously, we stated that security testing needs to accomplish two things: establish a shared reality of the security posture and automatically resolve any found issues. We'll explore both below.

Establish a shared reality about the security posture of the organization

Everyone, from the security engineers to the board of directors, should give the same answer when prompted to the question, how secure are we? Today, if you ask ten security engineers this question - inside the same organization - you'll get a skew of answers. And asking the chair of the board, you'll probably get a blank stare.

This is because security test results have historically required context from an engineer. Since all other testing strategies require an extrapolation step, engineers have been forced to guess at the resilience of the environment - something each engineer will do differently.

CST solves this by running tests at the true scale of the organization, meaning there is no extrapolation needed to provide a shared sense of reality.

Automatically resolve any security issues found

Manual resolution of security bugs is quickly becoming a legacy process. Today, when a bug is found from an assessment, an engineer will open a JIRA ticket and wait 6-months for the fix, then get called back in to validate the bug was squashed. This process will get harder and harder to maintain as A.I.-driven tools gain popularity and start finding more bugs than their human counterparts. Testing needs to both find issues and engage the defense on their resolution.

05 Conclusion

Main findings

Endpoints, or networked devices running code, represent the core of most environments. An endpoint is composed of an operating system and the applications or programs running on it, along with a defensive solution - either built into the endpoint or bolted on as a third-party application. Endpoints can be laptops, desktops, servers, cameras, televisions, IoT devices, traffic lights, thermostats, ... any device running code.

The purpose of endpoint defense (such as EDR) is to protect the operating system from malicious behaviors. Therefore, logic would dictate a security testing strategy should focus on the efficacy of the endpoint defense. There are other angles worth evaluating, such as packet inspection across the network or credential dumping followed by lateral movement, which are more informational than efficacy-related. Here, I've separated these into an advanced security posture best suited for manual red team assessments.

Continuous security testing is the state of active testing, at scale, to know with certainty if your defenses will protect you against emerging threats.

Application to the thesis

Continuous security testing should form the base of any security validation strategy.

If the endpoint is the center of the infrastructure universe, testing its defensive posture should be the base of any testing strategy. This argument was supported by examining the most competitive strategy today (BAS), comparing it to the methodology of CST, running an experiment around the effectiveness of continuous testing against popular defenses, and countering common arguments against autonomous testing.

Removing the human entirely - and leveraging the Law of Large Numbers to spray variable attack scenarios against endpoint defenses - provides CST with the most modern, and quantifiable, approach to security testing today.

Advancing your continuous security testing journey

Hopefully these 20+ pages have piqued your interest. If you can believe it, we've barely scratched the surface of CST. Stay curious and use this asset to raise the standard for control validation and hardening.

For readers that would like another 350+ pages of CST knowledge, consider picking up a copy Irreducibly Complex Systems at https://www.yellowduckpublishing.com.

If you're ready to bring continuous security testing to life, I invite you to create a free account for Prelude Detect - our production-scale continuous security testing platform. You'll get access to a library of Verified Security Tests, test authoring tools, and be able to scale testing to 25 endpoints - for free.

Create your FREE Prelude Detect account

About David Hunt

David Hunt is the co-founder and CTO of Prelude Security, the company pioneering production-scale continuous security testing. He leads engineering for Prelude, with responsibility over the security, engineering, and infrastructure teams.

Prior to Prelude, David was a group lead and principal security engineer at The MITRE Corporation, where he designed and built the CALDERA framework, an open-source tool for conducting semi-autonomous purple team assessments. David co-chaired MITRE's offensive security-related autonomous decision making research and served on the ATT&CK technical leadership team. David also led several initiatives with MITRE Engenuity, including involvement with participants in the Center for Threat Informed Defense (CTID).

Over his 17-year career, David has analyzed security across countless industries, including enterprise, aerospace, and OT. He is the author of three public and two private security testing solutions. David has worked for private and public organizations such as Mandiant, Kenna Security, Vodori, John Deere, Rockwell Collins, Vernon Research, and the U.S. government.

About Prelude Security

Prelude Security allows organizations to know with certainty if their defenses will protect them against the latest threats. Prelude Detect, the world's first production-scale continuous security testing platform, is designed for organizations of all sizes to continuously run security tests on production machines at scale. Leveraging Prelude's proprietary, kilobyte-sized processes called Probes, businesses can safely and continuously test how their defenses respond to adversarial behavior, CVEs, CISA Advisories, and more across their environments. The platform integrates with the leading defensive controls and feeds test efficacy data back to these solutions to create a self-healing defense. This allows customers to ensure their security infrastructure is properly configured to defend against critical threats. Prelude is backed by Sequoia Capital, Insight Partners, The MITRE Corporation, CrowdStrike Falcon Fund, IA Ventures, Four Rivers and other leading investors.

Create your FREE Prelude Detect account